Middleware Domain Technical Architecture

September 15, 2002

Version 2.0

History of Changes

Date	Modification
12/5/2000	Added a new section on XML and SOAP to Components. In Table 2 Middleware Product Selection Matrix added IBM WebSphere Host Publisher to Terminal Emulation Products – Strategic added STC e*Gate™ to Messaging and Application Integration Products – Research
11/22/2000	Removed table from page 8 that illustrated several types of messaging In Table 1 Middleware Standards Selection Matrix, changed OLE DB from transitional to strategic.
5/21/2002 **<NEW>**	In Table 2 Middleware Product Selection Matrix **removed** STC e*Gate™ from product list. In Table 2 Middleware Product Selection Matrix **added** Oracle 9iAS Application as a strategic product. In Table 2 Middleware Product Selection Matrix **changed** IBM MQSeries to preferred messaging middleware. Added JMS as preferred messaging middleware.

Technical Architecture for the Middleware Domain

Middleware Technical Architecture 1.0. doc

Mission Statement

Middleware architecture and products facilitate and simplify communication within and between services, application systems and components, whether distributed or not, or running on heterogeneous platforms (or not). The focus of this report is limited to application communication middleware and to data access middleware. It does not deal with network specific middleware or workflow middleware. These are covered in the Network Domain and Collaborative / Directory Services Domain architecture documents respectively.

Introduction and Background

As the state moves toward a more adaptable and open infrastructure, multiple applications must be able to exchange data across complex, heterogeneous environments. Therefore, the state's technical infrastructure must be able to deliver pertinent information at the right place and time and in a useful format. Middleware facilitates interchange of information in a distributed, multi-vendor, and heterogeneous systems environment while providing the same levels of security, reliability, and manageability traditionally associated with a monolithic, mainframe-based architecture where all products are supplied by a single vendor.

Middleware is software that supports communications between the functional tiers of an application, between two or more different applications, and between applications and shared services. The role of middleware is to insulate application developers from having to understand the complexities of the networking and computing environments and to prevent them from having direct interfacing to platform, network and data layers. Middleware also provides an environment in which to implement business rules (logic) and workflow rules.

Growth in Middleware

Middleware products are an evolving technology encompassing a wide range of capabilities from database access to very sophisticated integration engines known as message brokers. Much of the increased demand for middleware is due to several factors:

- The growth of Intra/Internet, data Mart and ERP applications greatly increased the need for sophisticated networked application architectures and integration of services. In particular these applications require integration with back-end legacy applications.
- Complexity and fragility of distributed platform infrastructure
- Component-based systems allowing applications to be "assembled" from re-usable parts (components)
- The rapid growth of Java as an object oriented language and now the increased use of XML (Extensible Markup Language) as the method of choice for cross platform data encoding, formatting and exchange.
- The intranet/internet explosion is fueling the demand for a new class of "human active" applications that require integration with back-end legacy applications;
- Message Oriented Middleware (MOM) products, are becoming increasingly popular. Once customers realize the benefits of simple one-to-one application connectivity with MOM, their interest in many-to-many application integration increases significantly.

- Recognition that a middleware solution needs to provide a rules-based engine that will manage the interface dialogues and control the workflow based on established business rules (and by implication the contents of the messages).
- Recognition that middleware products offer solutions to:
- Intra-application - Handles communication within the tiers of an application system.
- Inter-application - Handles communication between the application system and external services, such as common shared services and other application systems.
- Database Interface – Handles communication between application systems, or tiers, and database systems.

General Requirements for Middleware

- have a high level of reliability
- unpack or translate any data from whatever form it is delivered
- receive data from senders in the form they wish to send it in, using their preferred communication route
- determine, for any message, how to route it
- prevent sending specific types of data to receivers
- address the requirements of the organization in handling unavailable or error conditions encountered reformat and repackage data into forms intelligible to the intended receiver
- send data to the correct recipients in the form they wish to receive it, using their preferred communication route
- be aware of the availability of individual systems
- provide mechanisms to implement store and forward
- the capability to distribute messages, so that they do not need to be included in every application

Benefits of Using Middleware

By delivering common, standard solutions to cross-platform communication and related services, middleware products offer the following benefits.

Adaptability

A very important aspect of middleware is that by providing a common service through a standard application interface, applications are freed from a specific infrastructure. For example, applications requiring relational data can access that data from a DB2 database or an Oracle database just by changing the target DBMS, not the application. The underlying components of the technical infrastructure (such as operating systems, databases, and hardware platforms) can be expanded or changed without having to modify the application systems that are supported by the infrastructure.

Flexibility

The features and capabilities of applications can be modified without changes to the technical architecture. For the same reasons that the infrastructure can change without impacting the application, the application is free to change without necessarily impacting the infrastructure.

Reduced Development Effort

The use of middleware products simplifies the development of N-tier applications by providing common services for database access, transaction management and application-to-application communication. The logical partitioning of an *N*-tiered system and the modularity of construction using common services offer efficiencies through the specialization of skills and the reusability of components. These practices will improve the quality of systems and reduce lead times for their implementation and modification.

Reduced Integration Effort

Standardizing and expanding the capabilities offered through middleware products will allow purchased applications and services to be more easily integrated. A growing population of application packages is adopting industry middleware standards allowing these products to inter-operate.

Increased Return on Investment

- The opportunities for selecting products from different vendors are enhanced by the integration capabilities offered by middleware; therefore, greater competition will improve price offerings.
- Middleware can provides additional value-added business facilities such as auditing, common view of exposure, a single security platform, resource pooling and application accretion.
- In addition, implementation of good middleware solutions means that applications need to have only one interface to maintain and QA.

Components

XML and SOAP

Extensible Markup Language (XML) is a meta-markup language that provides a format for describing structured data. This facilitates more precise declarations of content and more meaningful search results across multiple platforms. XML has three important characteristics that will enable a new generation of Web-based data viewing and manipulation applications, and will enhance inter-and intra-application communications.

Simple Object Access Protocol (SOAP) provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC. Where XML allows very flexible encoding of data, SOAP defines a narrower set of rules for encoding.

XML Characteristics

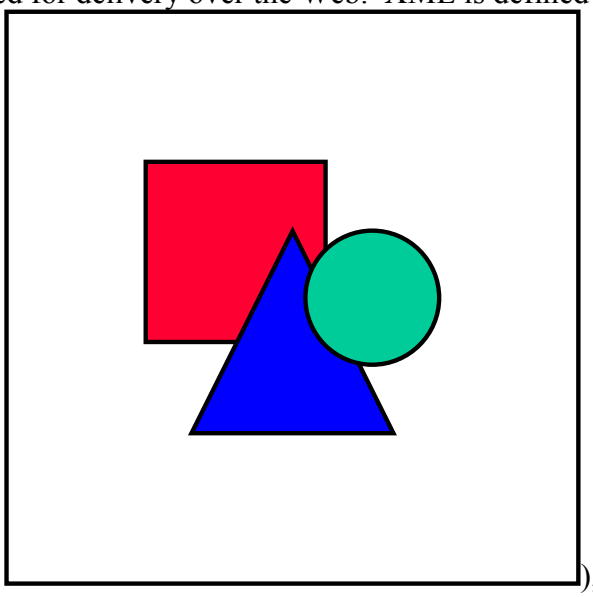
XML has three important characteristics that will enable a new generation of Web-based data viewing and manipulation applications, and will enhance inter-and intra-application communications.

XML is extensible

In XML, you can define an unlimited set of tags. While HTML tags can be used to display, a word in bold or italic, XML provides a framework for tagging structured data. An XML element can declare its associated data to be a retail price, a sales tax, a book title, the amount of precipitation, or any other desired data element. As XML tags are widely adopted, there will be a corresponding ability to search for and manipulate data, regardless of the applications within which it is found. Once data has been located, it can be delivered over a communications link and presented in a browser such as Internet Explorer 5 in any number of ways. Alternatively, the data can be handed off to other applications for further processing and viewing.

XML uses structural representation of data

XML provides a structural representation of data that has been broadly implemented and is relatively easy to deploy. XML is a subset of Standard Generalized Markup Language (SGML) that is optimized for delivery over the Web. XML is defined by the World Wide



Web Consortium (W3C), ensuring that structured data will be uniform and independent of applications or vendors.
Once the data is on the client desktop, it can be manipulated, edited, and presented in multiple views, without return trips to the server. Servers can now become more scalable, due to lower computational and bandwidth loads. In addition, since data is exchanged in the XML format, it can be easily merged from different sources.
XML is valuable to the Internet, as well as to large corporate Intranet environments because it provides interoperability using a flexible, open, standards-based format, with new ways of accessing legacy databases and delivering data to Web clients. Applications can be built more quickly, are easier to maintain, and can easily provide multiple views on the structured data.

XML separates data from the presentation and the process

The power and beauty of XML is that it maintains the separation of the user interface from the structured data. Hypertext Markup Language (HTML) specifies how to display data in a browser, XML defines the content. For example, in HTML, you use tags to tell the browser to display data as bold or italic; in XML, you only use tags to describe data, such as city

name, temperature, and barometric pressure. In XML, you use stylesheets such as Extensible Style Language (XSL) and Cascading Style Sheets (CSS) to present the data in a browser. XML separates the data from the presentation and the process, enabling you to display and process the data as you wish by applying different style sheets and applications.
This separation of data from presentation enables the seamless integration of data from diverse sources. Customer Information, purchase orders, research results, bill payments, medical records, catalog data, and other information can be converted to XML on the middle tier, allowing data to be exchanged online as easily as HTML pages display data today. Data encoded in XML can then be delivered over the Web to the desktop. No retrofitting is necessary for legacy information stored in mainframe databases or documents, and because HTTP is used to deliver XML over the wire, no changes are required for this function.

SOAP Characteristics

SOAP is a lightweight, XML based protocol for exchange of information in a decentralized, distributed environment. SOAP defines a mechanism to pass commands and parameters between HTTP clients and servers. SOAP doesn't care what operating system, programming language, or object model is being used on either the server side or the client side: it is utterly agnostic, except that it needs HTTP as a transport.

SOAP consists of three parts:

- The SOAP "envelope" construct defines an overall framework for expressing **what** is in a message, **who** should deal with it, and **whether** it is optional or mandatory.
- The SOAP "encoding rules" defines a serialization mechanism that can be used to exchange instances of application-defined datatypes; and
- The SOAP "RPC representation" defines a convention that can be used to represent remote procedure calls and responses.

In addition, the SOAP specification defines two protocol bindings that describe how a SOAP message can be carried in HTTP messages either with or without the HTTP Extension Framework (HTTP is the base protocol for the World Wide Web).

SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but as illustrated above, SOAP messages are often combined to implement patterns such as request/response.

SOAP implementations can be optimized to exploit the unique characteristics of particular network systems. For example, the HTTP protocol binding provides for SOAP response messages to be delivered as HTTP (HTML) responses, using the same connection as the inbound request. Regardless of the protocol to which SOAP is bound, messages are routed along a so-called "message path", which allows for processing at one or more intermediate nodes in addition to the ultimate destination.

Note: The combination of XML and SOAP has many parallels to the characteristics of EDI (Electronic Data Interchange). Like the EDI protocols and standards, XML and SOAP are the subject of industry wide efforts to bring consistency in definitions and naming conventions. More information on XML and SOAP can be found at W3C http://www.w3.org/TR/REC-xml, and at OASIS http://www.oasis-open.org/cover/soap.html.

Messaging-Oriented Middleware (MOM)

Messaging-oriented middleware, or more recently Loosely Coupled Message Passing (LCMP), provides applications with the ability to send and receive messages across platforms. The messages contain application-specified information and/or directives meaningful within the context of the application. The message is queued and made available to one or more target applications. One architecture for MOM is based on the "publish" (send) and "subscribe" (receive) paradigm. The leading MOM products are IBM's MQSeries and MS Message Queue.

Most MOM products include the following basic services for peer-to-peer communication:

- queuing messages in a permanent cache managed by the product
- guaranteeing the delivery of a message; guaranteed delivery eliminates complicated application logic ensuring that messages are received and processed.
- synchronous and asynchronous processing of messages. Asynchronous processing of queued messages frees the sender from waiting for a response. Synchronous processing requires the sender to suspend execution until results are sent back from the server.

Transaction Processing Monitors (TPM)

Transaction Processing Monitors (TPM) are middleware products servicing clients requiring transaction services in an n-Tier distributed application environment. TPM middleware products are important when applications require high transaction volumes, load balancing, failure recovery and fail-over capabilities.

TPM Services

Transaction Processing Monitors provide the following core services:

Transaction Integrity

Necessary services to ensure those atomic database transactions comprising a business transaction are applied successfully or not at all. If any one transaction fails, all transactions contained in the unit of work must be rolled back to return the database to its "before" state. If all succeed, all are committed to the database(s).

Two-Phase Commit

A means of implementing transaction integrity when there is more than one target database system (server) involved in the transaction. Two-phase commit ensures that all servers have successfully posted the transactions targeted at their database before committing these to the databases involved in the unit of work.

Failure Recovery

Failure Recovery provides means for reestablishing the appropriate connections and restarting transactions when network and platform outages occur.

Load Balancing

A feature of Transaction Monitors in which the server component manages the workload presented by the clients by fully utilizing the resources available. Transaction Monitors

generally utilize transaction priorities and multiple database sessions and/or threads to optimize throughput.

Object Oriented TPM

Microsoft Transaction Server and Sun's Java Transaction Service have extended their ORB middleware solutions to support Transaction Services. These become an alternative approach to providing an object based transaction monitor capability.

Microsoft Transaction Server

Microsoft® Transaction Server is a component-based transaction processing system for the development and deployment of server-centric applications built using Microsoft Component Object Model (COM) technologies.

Java Transaction Service

Java™ Transaction Service (JTS) specifies the implementation of a Transaction Manager which supports the Java Transaction API (JTA) specification and implements the Java mapping of the OMG Object Transaction Service (OTS) 1.1 specification. JTS uses the standard CORBA ORB/TS interfaces and Internet Inter-ORB Protocol (IIOP) for transaction context propagation between JTS Transaction Managers.
A JTS Transaction Manager provides transaction services to the parties involved in distributed transactions: the application server, the resource manager, the standalone transactional application, and the Communication Resource Manager (OMG CRM).

Object Request Brokers (ORBs)

Object Request Brokers provide the means for communicating between application objects (components) residing on different platforms. The OMG is a consortium of vendors and Corporations whose goal is to establish an industry-wide standard for object-to-object interoperability (**OMG -** www.omg.org)
The OMG defines the Object Request Broker (ORB) as: "an application framework providing interoperability between objects, built in (possibly) different languages, running on (possibly) different machines in heterogeneous distributed environments."

CORBA

CORBA (Common Object Request Broker Architecture) is an industry standard specification for communicating between distributed objects. The CORBA specification defines the OMG's Object Management Architecture, ORB facilities, interfaces, and supplementary services.

IIOP (Internet Inter-ORB Protocol) is the Web protocol developed by the OMG to implement CORBA solutions over the World Wide Web. IIOP enables browsers and servers to exchange integers, arrays, and objects that are more complex. (Standard HTML only supports transmission of text. XML (see above) is an extension to HTML that supports database access). Recently the JAVA community released a specification for RMI over IIOP, allowing SUN's RMI to access non-Java objects.

DCOM/DCOM+/DNA

DCOM+ and DNA represent Microsoft's middleware products for distributed components. DCOM offers similar capabilities to the CORBA specification, but differs from CORBA in the

way the connection between the application components is established and managed. DCOM+ defines the specifications and APIs for server side applets and objects. DCOM+/DNA is available only on Windows platforms (DNA specifically on Windows 2000). It does not support inter-operability between different CORBA objects; it only supports co-existence of CORBA and DCOM models.

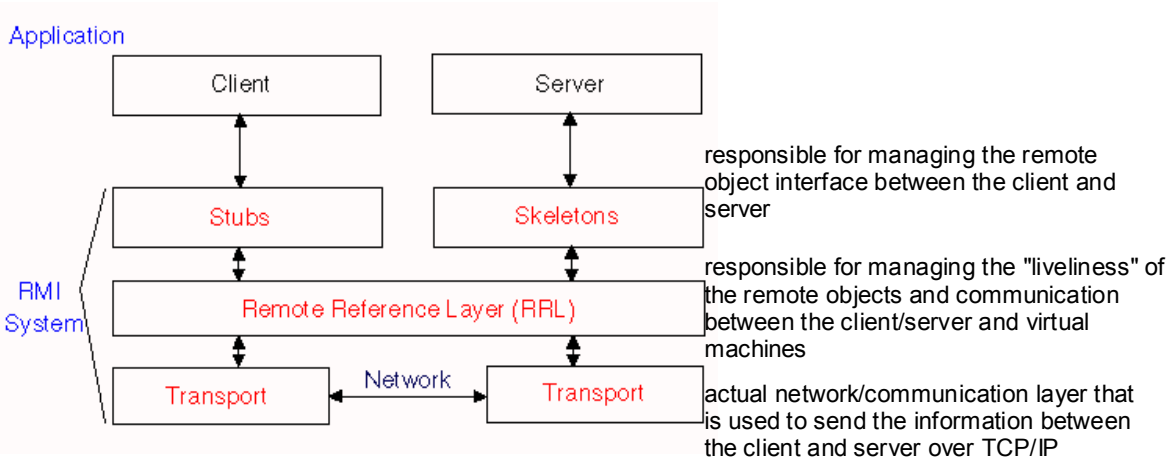
J2EE and EJB

Sun's Microsystems Java 2 Platform Enterprise Edition (J2EE) is a middleware specification intended to be implemented on "platform independent" Java Virtual Machines. EJB is the standard specification and APIs for server side applets and components. Key components of J2EE include RMI (below), JDBC and JTS. EJB interfaces well with IBM CICS, BEA Tuxedo, and ERP products such as SAP and PeopleSoft.

RMI

Remote Method Invocation, is a set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects. It is a true distributed computing application interface for Java. RMI is a relatively simple protocol that, currently, works only with Java objects. (unlike more complex protocols such as CORBA and DCOM)

Java RMI is comprised of three layers that support the interface (see illustration). When combined with the Java's Native Method Interface, RMI can connect to existing and legacy systems. RMI uses the Java Remote Method Protocol (JRMP) for interactions between distributed objects.



[Web reference material: http://java.sun.com/marketing/collateral/javarmi.html]

Database Middleware

Database middleware is the most prevalent form of middleware. It provides applications with the ability to access data stored in heterogeneous databases regardless of database management system and platform. The server-based middleware component, be it the DBMS server or Gateway server, is responsible for mapping the SQL requests to the DBMS-specific SQL, interfacing to the DBMS system and marshalling the result sets for transmission back to the

requestor. This includes data type conversions, caching of result sets, and packaging for transmittal.

There are two forms of database middleware product: client to DBMS server and client to gateway server. Both of these use a client-side driver to facilitate cross-platform communication to the server. Applications interface to the driver via an Application Programming Interface (API).

Client to Database Management Server

Oracle

NET8 (called SQL*NET prior to Oracle8) is Oracle's client/server middleware product that offers transparent connection from client tools to the database, or from one database to another. SQL*NET / Net8 works across multiple network protocols and operating systems. SQL*NET is a session level protocol under the OSI network model, to be present and properly functioning on both the client and the server.

Client to Gateway Server

Gateways provide bridging to multiple database management servers. Gateway's offer additional services and can be a way to manage DBMS workload. The features offered are often vendor-specific rather than standards-based such as the Sybase MDI product's Remote Stored Procedure (RSP), a form of Remote Procedure Call (RPC).

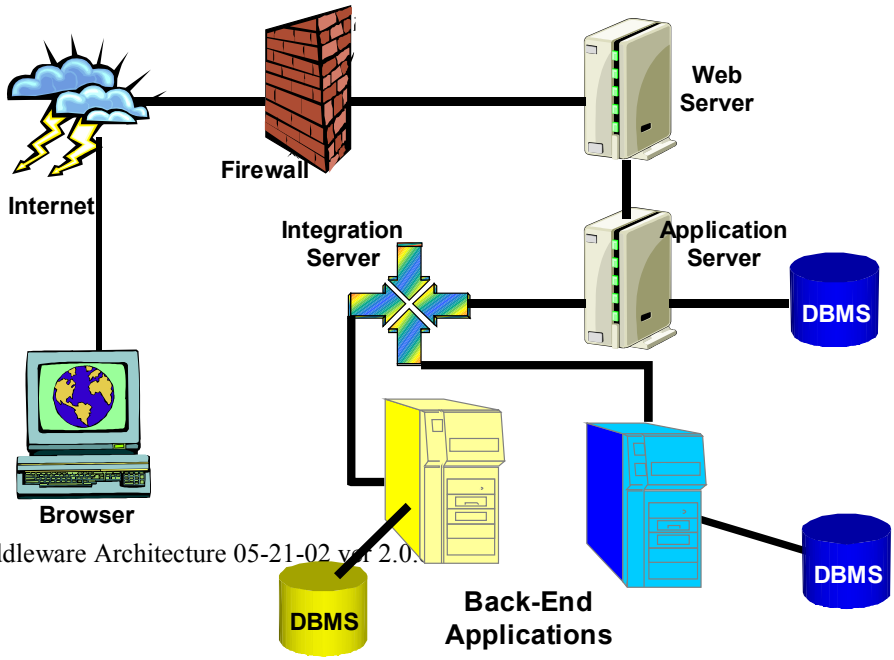
Application Programming Interfaces (APIs)

Industry Standard APIs

Microsoft's ODBC, provides a common API regardless of DBMS allowing easier migration and integration to various DBMS products. Two additional interfaces are JDBC for Java applications and OLE DB for Microsoft client applications.

Application/Integration Servers

The newest forms of middleware provides for the processing of business logic (application servers) and message based integration between applications (integration servers). The



distinction between application server and integration server is not always clear depending on the specific product offerings. This distinction will continue to be blurred over the next few years.

Application Servers

As access to business applications becomes more pervasive, it will drive responsive, adaptable client solutions. This will place additional demands on application servers to integrate content and produce multiple user interfaces based on common business logic. This will include the need to provide basic enterprise application integration and inter-enterprise integration facilities, along with workflow and a common user interface portal framework.

An essential facility provided by application serves is the execution of applications, (e.g., applets, components and objects) required to process business rules and actions, while insuring transactional integrity in a heterogeneous environment. To do so requires well designed application-programming interfaces (APIs) that are stable over time, present high-level abstraction of the services, and hide the details of the implementation. This in turn leads to a durable infrastructure that can be reused or expanded. A critical component of application execution will be workflow related, as this is the best way to capture the business model in executable form.

Another essential facility provided by application servers is the means for integrating existing data and applications or components (see Integration Servers below). Applications servers can provide the facilities that enable legacy and packaged applications to interact with each other, new Internet applications, and business partners. Application servers also offer functionality for supporting for naming and location, security, and higher messaging performance. The emerging XML standards (SOAP, ebXML, XML Schema) and UDDI initiatives will eventually provide the interoperability the various tools and servers will need to accomplish the integration and presentation tasks outline above

Integration Servers

Integration Servers are the latest generation of message brokers, which in turn evolved from data Mart transformation engines. This evolution allowed for the inclusion of other types of applications, such as those related to integrating ERP applications with other corporate systems.

An integration server is a software hub that records and manages the exchange of information between applications. The services provided by integration servers include data mapping, data transformations, event posting, process triggers and other means of automating data and process flow. Very often, the architecture employed is a variation of *publish* (send) and *subscribe* (receive). When a business event takes place, an application publishes the messages corresponding to that business event. The integration server reviews its lists of subscriptions and activates delivery to each subscriber for that message type. Subscribers receive only the data to which they subscribe. A message published by one application can be subscribed to by multiple consumer applications. Similarly, a subscribing application can receive messages from multiple publishing applications.

An important function of an integration server is to provide a business transaction coordination function (see Workflow Middleware below). A business transaction is typically made up of several units of work. Each unit of work must complete in order for the transaction to occur. If even one unit of work fails, the whole transaction fails, and all completed units of work must

then be reversed. These transactions are long running, and they require message-based updates to multiple databases.

The two most important integration server components are the rules-based engine and the data-transformation components. These two components work together to reconcile the differences between applications. Typically, development tools for these components provide a visual front end. This avoids the need for programming in a procedural language.

- The rules-based engine manages the interface dialogues and controls the workflow of the message stream. The business rules engine allows you to process messages based upon the unique requirements of the business and the processing of business activities. The rules-based engine need to make decisions on information contained in the messages. For example:
  - where to route messages
  - how to the identity the service required
  - how to manipulate the data
  - how to map the service requirement to the software components and systems that perform the required actions when to maintain synchronous interfaces when the ultimate service may be asynchronous
  - how to provide security and access to services through business rules
  - how to control, at the receiving interface, access to services, (e.g. is user allowed to make a payment of the value or type requested? Is the payment service closed?).
- The data transformation component is used to develop application-specific adapters. These adapters convert the data formats and applications semantics from the sending application to the receiving application.

Workflow Support

Middleware is a highly appropriate environment in which to manage workflow communications. It organizes the communication and exchange of data between the different applications that could be installed on various operating systems. Middleware also has a messaging system that can communicate with different applications as well as the ability to control the status of their processes. In addition, middleware adds intelligence in order to map data between the individual applications and the overall process (see Integration Servers above).

Interactions with Other Domains

The middleware domain is mostly interdependent on the principles, standards, best practices and product selections of the Application Development Domain and the Data Management / Data Warehouse Domain. In addition, there are some interdependencies with the Network, Platform and Collaboration / Directory Services domains.

Middleware Domain Principles

General Domain Principles

The general domain principles are based on the Conceptual Architecture Principles agreed to by the State of Connecticut. These general domain principles serve as a starting point for the evaluation, selection, acquisition, deployment and management of middleware technologies.

Principle 1. Consistency with other architectural principles.

Definition

The principles for decision making within this domain **must** also be filtered through these "standard domain" principles, in addition to the conceptual architecture principles

Rationale

As an "integration domain", this domain must interact with each of the principles across domain intersection points as well as support the over-arching conceptual architecture principles.

Principle 2. Use middleware to support logical partitioning and boundaries.

Definition

Messaging Oriented and Object Request Broker middleware technologies will be used to develop and maintain the logical partitioning of applications and databases within N-Tier architectures.

Rationale

Messaging Oriented and Object Request Broker middleware are the preferred methods to allow applications to be partitioned into discrete parts (objects) and distributed across multiple platforms. This is direct support of conceptual architectural principles 15 and 16.

Implications

- Requires a change in how applications are designed and built.
- Requires high degree of programming discipline to achieve real benefits.

Principle 3. Use mainstream technologies based on industry standards.

Definition

Middleware products and solutions will use industry-proven, mainstream technologies with priority given to products adhering to industry standards and open architecture. We will choose standards that are open, pervasive and non-proprietary, whenever possible.

Rationale

Industry standard protocols and interfaces minimize the dependence between application and platform infrastructure, and simplify the support of the distributed environment. Choosing proprietary systems, in lieu of open, may come at a higher cost to transfer to an open standard, or different standard, later. Using mainstream, standards based solutions

helps to assure adequate numbers of external implementers, training sources and technical support options.

Implications

- Need to have a mechanism to accommodate innovation from smaller, non-mainstream vendors.
- May mean replacement of current implementations sooner in their life cycle to achieve greater uniformity in deployment.

Principle 4. Use scalable middleware components and products.

Definition

The underlying middleware technology infrastructure and products must be scalable in size, capacity, and functionality to meet changing business and technical requirements.

Rationale

Use of scalable components and products encourages the re-use of those components and products. This reduces the overall cost of ownership.

Implications

- May increase the initial costs of development and deployment.
- Scalability must be reviewed for both upward and downward capability.

Principle 5. Minimize middleware configurations.

Definition

Middleware components and implementation will consist of a small number of standardized configurations that are designed for cross platform deployment and integration.

Rational

Reducing uniqueness in product selection and standardization reduces support and maintenance costs, and simplifies training and skills transfer. This is the most efficient approach to enterprise-wide middleware configuration and maintenance.

Implications

- May have to sub-optimize solutions in some business situations.
- Need to manage a process of regular replacement of middleware components to ensure the retirement of Not Supported and unique configurations.

Component Specific Principles

The component specific principles are, in essence, the “mandatory” guidelines to be followed by agencies or consultants when choosing or deploying middleware solutions or applications.

Principle 6. Use MOM and ORB middleware to reduce integration complexity.

Definition

Message Oriented Middleware (MOM) and Object Request Broker (ORB) middleware technologies will be the preferred method of reducing integration complexity when designing communication between the tiers of distributed applications, and between applications running on the same or different platforms.

Rationale

N-tier architecture is the preferred means of distributing applications (refer to Conceptual Architecture Principles). The tiers of a distributed application often run on different platforms (client, mid-tier application server, database server) and must be able to communicate. The use of MOM and ORB middleware helps to standardize the solutions to heterogeneous communication requirements thus reducing application complexity and total cost of ownership.

Implications

- Use of middleware for communication between application tiers and platforms may increase up-front development and acquisition costs.
- Requires a wider (possibly enterprise wide) focus when designing and deploying applications.
- Middleware components will have to designed, acquired, developed, or enhanced such that data and processes can be shared and integrated;

Principle 7. Use MOM when connecting to legacy systems.

Definition

Message Oriented Middleware should be the preferred method for "wrapping" legacy applications to provide connectivity between legacy systems and client/server or Internet applications. The legacy applications can be running on mainframes (or equivalents), older mid-range platforms.

Rationale

Message-oriented middleware is more appropriate to handle application to application communication when the target application is running on the mainframe. . MOM based solutions promote loosely coupled, highly granular solutions. This approach also yields greater flexibility and adaptability.

Implications

- Existing solutions such as "screen scraping" with single function products, or terminal emulation become "Not Supported" or transitional.
- The strategies for application integration will need to be fully evaluated against the current and future needs of the organization
- Advanced approaches such as legacy wrapping and message brokers require more effort to produce results than does screen scraping or using propriety database interface mechanisms.
- Places greater dependence on network or server based security and authorization mechanisms.

Principle 8. Use database middleware and TPM middleware.

Definition

A middleware layer will be used to connect the business logic tier to the data access tier (or back-end databases). This layer can be the database access components or transaction processing monitor (TPM) components.

Rationale

Using a database middleware layer buffers the application from platform dependent interfaces and SQL statements. Use of database middleware supports the adaptability of the systems architecture, and it supports the flexibility and scalability of applications using back-end databases.

Implications

- The facilities provided by transaction monitors will have to be qualified against the application requirements to determine if such an approach is warranted.
- Investment in this technology requires systems management expertise to configure and monitor usage.
- Proprietary database middleware should be implemented in a manner designed to reduce product dependencies.

Principle 9. Use asynchronous communications.

Definition

Asynchronous messaging (MOM, ORBs etc.) products and configurations should be used when ever possible. Synchronous communication will be used if appropriate in the context of the applications; even then, preference will be given to asynchronous communication run in pseudo-synchronous mode.

Rationale

Asynchronous communication offers more flexibility than synchronous communication. Asynchronous messaging allows the downstream process to decide on an appropriate processing strategy to optimize its throughput and/or respond to different priority requests. This promotes increased performance, flexibility and scalability of the application.

Implications

- Requires changes in the application developer's "mind-set".

Principle 10. Use MOM when providing connectivity for remote users.

Definition

Message-oriented middleware is appropriate when remote processes need to post to post transactions on central servers after connecting via dial up facilities or other bandwidth limited communication links.

Rationale

The transactions can be queued and asynchronously processed by a target application. This use of MOM takes advantage of asynchronous processing and guaranteed delivery

capabilities. In addition, message queuing allows messages to be stored by the middleware and acted upon when the receiving application chooses to.

Implications

- Requires that security be a service that can be used by middleware components.

Principle 11. Consistency In Naming Conventions

Definition

Middleware components and objects will utilize a consistent naming convention and schema.

Rationale

Consistency in naming conventions and schema will facilitate use and re-use of components, thus increasing productivity and uniformity.

Implications

- Achieving consistency in naming conventions and schema will require considerable planning, and collaboration between business units. This is not easy to achieve.
- There are considerable dependencies on the activities in the network and collaborative/directory for achieving consistency in naming.

Principle 12. Reuse Security and Access Control Components.

Definition

Middleware components will not duplicate identification, authorization or access controls already provided by common or shared security or access control mechanisms or servers.

Rationale

Reusing the common security and access control mechanisms avoids inconsistencies and redundant work needed to provide and maintain those mechanisms. This principle was called out explicitly to denote the importance of the interaction between this domain and the Network and Collaborative/Directory domains. This principle is consistent with the intent of the principles in both those domains.

Implications

- Middleware components act as “conduits” for security and access control information. Authentication and authorization are a function of other network or infrastructure services.

Best Practices Principles

The "best practices" principles are intended to ensure that the State achieves a high degree of return on its investments in middleware technology.

Principle 13. Availability of training and technical support.

Definition

Training and technical support must be available for all middleware tools and software selected. A program of regular training will be provided to developers and designers in the effective use of middleware tools, software and components. In addition, technical support must be made available either through internal resources, or through "outsourcing", or through both.

Rationale

The correct and effective implementation of middleware components requires knowledgeable personnel. A lack of adequate technical support often leads to failed projects or an inability to provide timely problem resolution.

Implications

- This training will be targeted to those who need such training
- The appropriate timing for training will have to be factored into the development of projects and systems.
- Requirements for technical support must be included in project definitions and in procurements when necessary.

Principle 14. Document object functionality and public interfaces.

Definition

When using object oriented middleware, developers will document public interfaces and object functionality. This is also applicable to the components of message oriented middle implementations.

Rationale

The documentation of public interfaces and object functionality will facilitate sharing of components across applications and the reuse of components.

Implications

- Will need to use the facilities of the language or tool to automate this activity.
- Developers will need to provide example code or applets as part of the documentation.

Standards

The purpose of an EWTA is to create a business-driven blueprint for the application of technology toward solving business problems. Effective architectures are prescriptive. The domain technical architecture must guide and direct infrastructure and development engineering decisions. Standards are an important vehicle for providing this guidance and play an important role in enabling easy interchange of components from multiple vendors. The

primary value, offered by IT industry standards, is to enable integration of systems and applications both within and between enterprises.

There are two types of standards that are addressed in this paper:

- Technical standards, to which products or implementations must comply with, or conform to, and
- Product Standards, specific vendor offerings that the State has selected from among the products that comply with, or conform to, the technical standards.

Life Cycle

Standards, whether technical or product have a life cycle which encompasses four major categories:

Obsolete Standards

It is highly likely that these standards or products, while still in use, will not be supported by the vendor (industry, manufacturer, etc.) in the future. Some products and standards have already reached the non-supported state. Plans should be developed by the agencies or the State to rapidly phase out and replace them with strategic standards or products. No development should be undertaken using these standards or products by either the agencies or the State.

Transitional Standards

These are standards or products in which an agency or the State has a substantial investment or deployment. These standards and products are currently supported by DOIT, the agencies, or the vendor (industry, manufacturer, etc.). However, agencies should undertake development using these standards or products only if there are no suitable alternatives that are categorized as strategic. Plans should be developed by the agencies or the State to move from transitional to strategic standards or products as soon as practical. In addition, the State should not use these standards or products for development.
Note: many older versions of *strategic* standards or products fall into this category, even if not specifically listed in a domain architecture document.

Strategic Standards

These are the standards and products selected by the state for development or acquisition, and for replacement of *obsolete* or *transitional* standards or products. (Strategic means a three to four year planning horizon.) When more than one similar strategic standard or product is specified for a technology category, there may be a preference for use in statewide or multi-agency development. These preferred standards and products are indicated where appropriate.
Note: some strategic products may be in "pilot testing" evaluation to determine implementation issues and guidelines. Pilot testing must be successfully completed prior to full deployment by the agencies or the State.

Research / Emerging Standards

This category represents proposed strategic standards and products that are in advanced stages of development and that should be evaluated by the State. The some of these standards or products may already be undergoing "hands-on" evaluation. Others will need to be tracked and evaluated over the next 6 to 18 months.

Technical Standards

The technical standards chosen by the domain team were evaluated as:

- to their general compliance or non-compliance with the enterprise and domain architectures
- the relationship to existing standards here in the State of Connecticut, and
- to general technology directions. Preferences were given to standards that are sponsored by standards bodies, followed by standards enjoying wide market support (de facto) . Least preference was given to proprietary standards.

The following standards apply to interfaces and transport mechanisms.

Table 1 Middleware Standards Selection Matrix below summarizes the technical middleware architecture as the State of Connecticut moves towards its future environment

Current and Strategic Standards

Standard 1: Microsoft's ODBC – database access API

Standard 2: Sun's JDBC – database access API.

Standard 3: IBM MQSeries Application Messaging Interface (AMI) – adopted by the *Open Applications Group*.

Standard 4: Microsoft Message Queue (MSMQ) –interface API.

Standard 5: EDI (Electronic Data Interchange) – cross platform business data encoding and formatting, and interchange protocol.

Standard 6: XML (Extensible Markup Language) – cross platform data encoding and formatting.

Emerging Standards

Emerging 1: SOAP (Simple Open Access Protocol) – for invoking distributed system services or making remote procedure calls.

Emerging 2: UDDI (Universal Discovery Description And Integration) – specifications for publishing and discover information about web services.

The following standards apply to Object Based middleware.

Current or Strategic Standards

Standard 7: CORBA's Interface Definition Language (IDL) for defining public interfaces, and Transaction Services (TS) interface definition.

Standard 8: OMG Object Transaction Service (OTS) 1.1 –interfaces and protocol specifications for CORBA based transaction services.

Standard 9: RMI (Remote Method Invocation) – enables one Java application to access the objects and methods of another Java application.

Standard 10: RMI over IIOP (Remote Method Invocation over Internet Inter-ORB Protocol) – enables one Java application to access the objects and methods of another Java or CORBA program across the Internet.

Standard 11: JTS (Java Transaction Service) –

Standard 12: Sun Microsystems J2EE and EJB – overall architecture for APIs, protocols and server-side s Java components such as Java Beans or Enterprise Java Beans (EJBs) encapsulating application logic.

Standard 13: Microsoft DNA/DCOM+ – overall architecture for APIs, protocols and server-side components using Microsoft's WIN NT, WIN2K based operating environments. This standard subsumes OLE DB.

Emerging Standards

Emerging 3: Java Message Service (JMS) API – common API and provider framework for , asynchronous communication between components in a distributed computing environment.

Emerging 4: JDOM (Java Document Object Model) – creates a new set of Java classes and interfaces for manipulating XML documents; it is optimized for Java.

Emerging 5: Microsoft .NET – programming model for creating XML-based Web Services and core, .NET building block services. (This proprietary environment is due out in the 2002 time frame.)

Middleware Product Standards

The domain team selected product standards on the basis of 4 criteria:

1. the fit with Conceptual Architectural Principles,
2. the fit with the Requirements for Technical Architecture,
3. the fit with the domain specific principles identified above, and
4. an analysis of the current trends in technology and the market place.

Table 2 Middleware Product Selection Matrix below summarizes the product direction as the State of Connecticut moves towards its future environment.

Table 1 Middleware Standards Selection Matrix

	Status Category			
Standard Existing or Proposed	**Obsolete**	**Transitional**	**Strategic**	**Research**
Interface / Transport				
ODBC			✔	
JBDC			✔	
OLE		✔		
OLE DB			✔	
MSMQ API			✔	
AMI (IBM)			✔	
OMG/CORBA			✔	
SOAP				✔
XML			✔	
EDI			✔	
UDDI				✔
Object Oriented				
IDL (OMG)			✔	
CORBA/CCM			✔	
OTS1.1 (OMG)			✔	
RMI (Sun)			✔	
RMI over IIOP (Sun)				✔
JTS			✔	
EJB			✔	
J2EE			✔	
JMS API (J2EE 1.3)			✔	
JDOM				✔
DNA/DCOM+ (Microsoft)			✔	
.NET (Microsoft)				✔

Table 2 Middleware Product Selection Matrix

Product Existing or Proposed	Obsolete	Transitional	Strategic	Research
TP Monitors				
IBM TX Series (CICS)			✔	
Microsoft MTS			✔	
JTS (SUN)				✔
Terminal Emulation				
Screen Scappers		✔		
IBM Host on Demand			✔	
IBM WebSphere Host Publisher			✔	
IBM DB2 Connect			✓	
Messaging				
IBM's MQSeries **[changed]**			✔ preferred	
JMS (SUN)			✔ preferred	
Microsoft's MSMQ			✔	
Application Integration				
Oracle 9iAS Application Server **[NEW]**			✔ preferred	
IBM WebSphere			✔ preferred	
IBM MQSI			✔	
.NET (Microsoft)			✔	
DBMS Middleware				
Oracle *Net (*Net)		✔		